

Module 8 — Outils Avancés

Jour 2 — Ralph, sprints autonomes et intégrations

The Bearded Bear

Février 2026



Table des matières

Module 8 : Outils Avancés, Ralph et Autonomie	3
Objectifs	3
1. Systeme de Hooks (15min)	3
Qu'est-ce qu'un Hook?	3
Les 13 evenements hooks	3
Configuration dans settings.json	4
Regles importantes	4
Matchers specifiques par evenement	5
Prompt-based hooks (type : "prompt")	5
Hook scripts Claude-Craft 7.13.0	5
Exemple : Hook de lint automatique PHP	6
Exemple : Hook de securite (bloquer des operations)	6
2. Ralph Wiggum - Boucle IA Continue (15min)	7
Qu'est-ce que Ralph?	7
Lancement	7
Validateurs Definition of Done (DoD)	7
Circuit Breaker adaptatif	7
Dashboard temps reel	8
Configuration (ralph.yml)	8
3. Sprint Automation with Ralph + BMAD (15min)	9
Sprint Development avec Ralph et BMAD	9
Approche recommandee	9
Agent Teams pour sprints paralleles (v2.1.32+)	9
Bonnes pratiques	9
4. Extended Thinking et MCP (15min)	10
Extended Thinking	10
MCP (Model Context Protocol)	10
5. QA Recette - Tests d'Acceptance Automatisés (15min)	12
La Regle d'Or	12
Prerequis	12
Commandes	12
Comment ca fonctionne	13
Categories de tests generes	13
Structure de sortie	13
Session Recovery	13
6. Permissions 3-tier et Plugins (15min)	14
Systeme de permissions	14
Configuration dans settings.json	14
Wildcards supportes	14
Niveaux de configuration	15
Systeme de plugins	15

7. Task Management System (10min)	16
Qu'est-ce que le Task Management?	16
Les 4 outils	16
Cycle de vie (v2.1.20+)	16
Task Tool Metrics (v2.1.30+)	16
Gestion des dependances	17
Cas d'usage	17
8. File Operation Tools vs Bash (5min)	17
Pourquoi preferer les outils natifs?	17
Mapping des outils	18
PDF Page Range Support (v2.1.30+)	18
Quand utiliser Bash?	18
9. PR Integration (5min)	19
Qu'est-ce que PR Integration?	19
Utilisation	19
Auto-link	19
Indicateurs de statut	19
10. Background Agents et Permissions (5min)	19
Permission Prompting (v2.1.20+)	19
Comportement	20
Options de reponse	20
Avantages	20
12. Personnalisation avancee (5min)	20
spinnerVerbs (v2.1.23+)	20
Settings Enhancement (v7.13.0)	21
11. Agent Teams (v2.1.32+ Research Preview) (10min)	21
Qu'est-ce qu'Agent Teams?	21
Activation	21
Outils disponibles	21
Fonctionnement	22
Modes d'affichage	22
Cas d'usage	22
Claude Opus 4.6 (v2.1.32+)	22
Points Cles a Retenir	23

Module 8 : Outils Avances, Ralph et Autonomie

Objectifs

A la fin de ce module, vous serez capable de : - Configurer et utiliser le systeme de Hooks (13 evenements) - Maitriser Ralph Wiggum pour la boucle IA continue - Automatiser les sprints avec Ralph, BMAD et Agent Teams - Utiliser Extended Thinking et MCP - Exploiter QA Recette pour les tests d'acceptance - Configurer les permissions et plugins - Comprendre Agent Teams et la coordination multi-agents

1. Systeme de Hooks (15min)

Qu'est-ce qu'un Hook ?

Les **Hooks** sont des commandes shell qui s'executent automatiquement lors d'evenements specifiques dans Claude Code. Ils permettent d'automatiser des workflows, d'ajouter des validations et d'integrer des outils externes.

Les 13 evenements hooks

Attention : Les evenements PreWrite, PostWrite, PreBash, PostBash, Error et TokenThreshold que l'on trouve dans certaines documentations sont **faux**. Voici les 13 evenements reellement supportes (source : <https://code.claude.com/docs/en/hooks>) :

Event	Declencheur	Usage typique
PreToolUse	Avant execution d'un outil	Validation, confirmation, backup
PostToolUse	Apres succes d'un outil	Lint, format, notification
PostToolUseFailure	Apres echec d'un outil	Auto-recovery, logging d'erreurs
PermissionRequest	Demande de permission	Controle d'accès automatise
UserPromptSubmission	Soumission prompt utilisateur	Transformation, validation input
Stop	Fin de reponse Claude	Cleanup, rapport de session
SubagentStop	Fin d'un sous-agent	Agregation resultats
SubagentStart	Lancement d'un sous-agent	Logging, configuration contexte
Notification	Conditions d'alerte	Alertes Slack/Teams
PreCompact	Avant compaction contexte	Sauvegarde etat, checkpoint
SessionStart	Debut/reprise de session	Initialisation environnement
SessionEnd	Fin de session	Cleanup, statistiques

Event	Declencheur	Usage typique
Setup	Premier lancement (-init, -maintenance)	Installation dependances

Configuration dans settings.json

Les hooks se configurent dans `.claude/settings.json` :

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Write",
        "command": "echo 'Fichier va etre modifie'"
      }
    ],
    "PostToolUse": [
      {
        "matcher": "Write",
        "command": "docker compose exec app vendor/bin/php-cs-fixer fix
↪ --quiet"
      }
    ],
    "Stop": [
      {
        "command": "echo 'Session terminee'"
      }
    ],
    "Notification": [
      {
        "command": "curl -X POST -H 'Content-type: application/json' --data
↪ '{\"text\": \"Alerte Claude\"}' $SLACK_WEBHOOK_URL"
      }
    ]
  }
}
```

Regles importantes

Regle	Detail
Timeout	10 minutes maximum par hook
stdout	Affiche dans le contexte de Claude (Claude le voit)
stderr	Affiche a l'utilisateur (Claude ne le voit pas)

Regle	Detail
Exit code 0	Succes, l'action continue
Exit code 2	Bloque l'action (PreToolUse, UserPromptSubmit)
Autre exit code	Erreur non-bloquante (logged mais n'arrete pas)
Matcher	Filtre sur le nom de l'outil (ex : <code>Write</code> , <code>Bash</code> , <code>Edit</code>)

Matchers spécifiques par evenement

Certains evenements disposent de matchers specialises :

Event	Matchers disponibles
Notification	<code>permission_prompt</code> , <code>idle_prompt</code> , <code>auth_success</code> , <code>elicitation_dialog</code>
PreCompact	<code>manual</code> , <code>auto</code>
SessionStart	<code>startup</code> , <code>resume</code> , <code>clear</code> , <code>compact</code>
Setup	<code>init</code> , <code>maintenance</code>

Prompt-based hooks (type : “prompt”)

En plus des hooks shell classiques, Claude Code supporte les **prompt-based hooks** qui utilisent un LLM (Haiku) pour prendre des decisions contextuelles :

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "type": "prompt",
        "prompt": "Verify this command is safe and doesn't delete important
                  ↪ files"
      }
    ]
  }
}
```

Le LLM Haiku evalue le contexte et decide si l'action doit etre autorisee ou bloquee, offrant une validation plus intelligente qu'un simple script shell.

Hook scripts Claude-Craft 7.13.0

Claude-Craft 7.13.0 fournit 3 scripts de hooks pre-configures :

Script	Evenement	Description
post-tool-failure.sh	PostToolUseFailure	Auto-recovery apres echec d'un outil
pre-compact.sh	PreCompact	Backup de l'etat du sprint avant compaction
session-end.sh	SessionEnd	Collecte de metriques et cleanup de session

Exemple : Hook de lint automatique PHP

```
{
  "hooks": {
    "PostToolUse": [
      {
        "matcher": "Write",
        "command": "docker compose exec app vendor/bin/php-cs-fixer fix
↪ --quiet 2>/dev/null"
      },
      {
        "matcher": "Write",
        "command": "docker compose exec app vendor/bin/phpstan analyse
↪ --level=8 --no-progress 2>/dev/null"
      }
    ]
  }
}
```

Exemple : Hook de securite (bloquer des operations)

```
{
  "hooks": {
    "PreToolUse": [
      {
        "matcher": "Bash",
        "command": "echo 'Validation de la commande avant execution' >&2"
      }
    ]
  }
}
```

2. Ralph Wiggum - Boucle IA Continue (15min)

Qu'est-ce que Ralph ?

Ralph Wiggum est un système de boucle continue qui exécute Claude de manière répétée jusqu'à ce qu'une tâche soit terminée. Il orchestre les iterations, vérifie les critères de completion (Definition of Done), et gère les erreurs automatiquement.

Lancement

```
# Utilisation basique
/common:ralph-run "Implementer authentication utilisateur"

# Detection automatique du projet
/common:ralph-run "Corriger le bug de connexion" --auto-detect

# Mode interactif pour configurer
/common:ralph-run --interactive

# Reprendre une session
/common:ralph-run --continue=ralph-1704067200-a1b2
```

Validateurs Definition of Done (DoD)

Le système DoD vérifie la completion via plusieurs types de validateurs :

Type	Description	Exemple
command	Executer commande shell (tests, lint, build)	docker compose exec app vendor/bin/phpunit
output_contains	Vérifier pattern dans sortie Claude	<promise>COMPLETE</promise>
file_changed	Vérifier modification de fichiers	src/Entity/User.php
hook	Intégrer avec quality-gate.sh	Hook Claude existant
human	Validation humaine interactive	Approbation manuelle

Circuit Breaker adaptatif

Le circuit breaker empêche les boucles infinies en adaptant les seuils selon le type de tâche :

Profil	Mots-cles	Sans Modif	Erreurs	Max Iter
quick_fix	fix, bug, typo	2	3	10
small_feature	add, implement	3	4	15
medium_feature	feature, create	4	6	25
large_feature	refactor, migrate	5	8	50
exploration	explore, investigate	10	15	100

Dashboard temps reel

```

+=====+
|  RALPH WIGGUM - Session: ralph-xxx          PHASE: GREEN          |
+=====+
|  ITERATION 8/25                      ECOULE: 12:34                |
|  PROGRES #####..... 32%          |
|                                     |
|  Circuit Breaker: .. (0/4)    Contexte: #####.. 78%          |
+=====+

```

Configuration (ralph.yml)

```

version: "2.0"

hooks:
  enabled: true
  mode: "advanced"

auto_detect:
  enabled: true

dashboard:
  enabled: true
  mode: "full"

circuit_breaker:
  adaptive: true
  default_profile: "medium_feature"

definition_of_done:
  checklist:
    - id: tests
      type: command
      command: "docker compose exec app vendor/bin/phpunit"
      required: true

```

```
- id: completion
  type: output_contains
  pattern: "<promise>COMPLETE</promise>"
  required: true
```

3. Sprint Automation with Ralph + BMAD (15min)

Sprint Development avec Ralph et BMAD

Depuis la v7.0.0, l'automatisation de sprints se fait via la combinaison de **Ralph Wiggum** (boucle IA continue) et des **commandes BMAD** (/sprint:*, /project:*). L'ancien ASC (Autonomous Sprint Conductor) et /common:ralph-sprint ont été retirés en v6.0.0.

Approche recommandée

```
# 1. Executer un sprint complet via les commandes projet
/project:run-sprint

# 2. Ou utiliser Ralph pour une story spécifique
/common:ralph-run "Implement US-042: Add email validation"

# 3. Sprint parallèle avec Agent Teams
/team:sprint
```

Agent Teams pour sprints parallèles (v2.1.32+)

Avec Agent Teams, plusieurs agents peuvent travailler sur des stories indépendantes en parallèle :

```
# Activer Agent Teams
export CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS=1

# Lancer un sprint parallèle via la commande team
/team:sprint
```

Bonnes pratiques

1. **Utiliser les commandes BMAD** : /sprint:next-story, /sprint:transition, /gate:validate-story
 2. **Ralph pour les stories individuelles** : /common:ralph-run avec DoD validators
 3. **Agent Teams pour le parallélisme** : /team:sprint pour traiter plusieurs stories
 4. **Quality gates** : Toujours valider avec /gate:validate-story avant de transitionner
-

4. Extended Thinking et MCP (15min)

Extended Thinking

L'Extended Thinking permet de demander à Claude de réfléchir plus profondément avant de répondre. Il existe une hiérarchie de niveaux :

Niveau	Commande	Profondeur	Usage
Basique	<code>think</code>	Reflexion standard	Questions simples
Approfondi	<code>think hard</code>	Reflexion approfondie	Problemes moderelement complexes
Intense	<code>think harder</code>	Reflexion intense	Problemes complexes
Maximum	<code>ultrathink</code>	Reflexion maximale	Architecture, debug multi-fichiers

Utilisation Il suffit de prepender le mot-cle à n'importe quel prompt :

`think hard` Comment restructurer ce module en Clean Architecture ?

`ultrathink` Analyse ce bug de concurrence dans le systeme de paiement

Quand utiliser chaque niveau

Situation	Niveau recommande
Correction de typo, petit fix	Pas necessaire
Nouvelle feature standard	<code>think</code>
Architecture multi-modules	<code>think hard</code>
Debug complexe multi-fichiers	<code>think harder</code>
Decision architecturale critique	<code>ultrathink</code>

MCP (Model Context Protocol)

Le **Model Context Protocol** (MCP) est un protocole ouvert pour connecter des outils externes à Claude Code. Il permet d'étendre les capacités de Claude avec des serveurs specialises.

Gestion des serveurs MCP

Commande de gestion

`/mcp`

Variable d'environnement pour le timeout

`export MCP_TIMEOUT=30000` *# 30 secondes*

Cas d'usage MCP

Serveur MCP	Capacite ajoutee
Base de donnees	Requetes SQL directes
API externe	Appels REST/GraphQL
Navigateur	Automatisation web (Chrome)
Systeme de fichiers	Acces etendu
Monitoring	Metriques temps reel

OAuth Client Credentials (v2.1.30+) Pour les serveurs MCP qui ne supportent pas le Dynamic Client Registration :

```
claude mcp add --client-id YOUR_CLIENT_ID --client-secret YOUR_CLIENT_SECRET  
↪ slack -- npx -y @modelcontextprotocol/server-slack
```

Flag	Description
<code>--client-id</code>	Client ID OAuth pour le serveur MCP
<code>--client-secret</code>	Client secret OAuth pour le serveur MCP

Configuration MCP Les serveurs MCP se configurent dans `.claude/settings.json` :

```
{  
  "mcpServers": {  
    "database": {  
      "command": "mcp-server-sqlite",  
      "args": ["--db", "app.db"]  
    },  
    "browser": {  
      "command": "mcp-server-chrome",  
      "args": []  
    }  
  }  
}
```

5. QA Recette - Tests d'Acceptance Automatisés (15min)

La Regle d'Or

Un bug corrige ne doit JAMAIS reapparaître.

C'est le principe fondamental de QA Recette. Chaque erreur detectee genere automatiquement un test de regression qui sera rejoue a chaque recette future.

Prerequis

- Extension Chrome Claude in Chrome v1.0.36+
- Claude Code lance avec `--chrome` ou `/chrome`

Commandes

```
# Tester une story specifique
/qa:recette --scope=story --id=US-001

# Tester un sprint entier
/qa:recette --scope=sprint --id=Sprint-3

# Dry run (voir le plan sans executer)
/qa:recette --scope=story --id=US-001 --dry-run

# Reprendre une session interrompue
/qa:recette --resume=REC-20260130-143022

# Voir le statut d'une session
/qa:status

# Consulter les tests de regression
/qa:regression

# Generer un rapport
/qa:report --session=REC-xxx --format=markdown

# Corriger les bugs d'une session recette
/qa:fix --session=REC-20260130-143022

# Dry run : affiner et documenter sans corriger
/qa:fix --session=REC-20260130-143022 --dry-run

# Corriger uniquement les bugs critiques
/qa:fix --session=REC-20260130-143022 --severity=critical

# Generer les documents BMAD sans lancer le TDD
/qa:fix --session=REC-20260130-143022 --skip-fix
```

Comment ça fonctionne

1. **Generation du plan** : QA Recette lit les criteres d'acceptance de la story et genere un plan de test complet
2. **Execution via Chrome** : Les tests sont executes dans un vrai navigateur via Claude in Chrome
3. **Classification des erreurs** : Chaque erreur est classifiee (visuelle, interaction, validation, logique, securite, API)
4. **Generation de regression** : Pour chaque erreur trouvee, un test de regression est automatiquement genere
5. **Rapport** : Un rapport detaille est produit avec tracabilite
6. **Correction TDD** : /qa : fix analyse les erreurs, genere des bug stories BMAD, et corrige chaque bug via le workflow TDD (RED → GREEN → REFACTOR)

Categories de tests generes

Categorie	Priorite	Description
Validation criteres acceptance	Critique	Chaque critere AC teste individuellement
Cas limites	Haute	Conditions aux bornes
Scenarios d'erreur	Haute	Gestion des erreurs et recuperation
Verification UI/UX	Moyenne	Coherence et utilisabilite
Checks performance	Moyenne	Temps de chargement et reponse
Securite basique	Haute	XSS, CSRF, injection

Structure de sortie

```
.recette/
  plans/           # Plans de test (YAML)
  sessions/        # Etats de session (checkpoints)
  regression/      # Suite de regression
    registry.yaml  # Registre de tous les tests
    tests/         # Code des tests generes
  metrics/         # Donnees historiques
  reports/         # Rapports generes
```

Session Recovery

QA Recette cree des checkpoints a chaque test. Si une session est interrompue, elle peut etre reprise exactement ou elle s'etait arretee :

```
# Reprendre une session interrompue
/qa:recette --resume=REC-20260130-143022
```

6. Permissions 3-tier et Plugins (15min)

Système de permissions

Claude Code utilise un système de permissions à 3 niveaux, appliqués dans cet ordre de priorité :

Niveau	Comportement	Priorité
Deny	Interdit complètement l'action	Plus haute
Allow	Autorise sans confirmation	Moyenne
Ask	Demande confirmation à chaque fois	Plus basse (default)

Configuration dans settings.json

```
{
  "permissions": {
    "allow": [
      "Read",
      "Glob",
      "Grep",
      "Write(src/**)",
      "Edit(src/**)",
      "Bash(docker compose *)",
      "Bash(*--help*)",
      "Bash(*-h*)"
    ],
    "deny": [
      "Bash(rm -rf *)",
      "Bash(git push --force*)",
      "Write(.env*)"
    ]
  }
}
```

Wildcards supportées

Pattern	Description	Exemple
*	N'importe quelle chaine	Bash(*-h*)
**	Recuratif dans les chemins	Write(src/**)
Nom outil seul	Toutes utilisations	Read
Outil(pattern)	Usage filtre	Bash(docker compose *)

Niveaux de configuration

Les permissions se definissent a plusieurs niveaux, du plus specifique au plus general :

1. **Projet** : `.claude/settings.json` (dans le repo)
2. **Utilisateur** : `~/ .claude/settings.json` (personnel)
3. **Entreprise** : `/etc/claude/settings.json` (deploiement entreprise)

Le niveau le plus specifique l'emporte.

Systeme de plugins

Claude Code peut etre etendu via des plugins qui ajoutent de nouvelles commandes et agents.

Structure d'un plugin

```
.claude-plugin/
  plugin.json      # Manifeste du plugin
  commands/        # Commandes ajoutees
  agents/          # Agents specialises
  hooks/           # Scripts de hooks
```

Exemple de manifeste plugin.json

```
{
  "name": "mon-plugin",
  "version": "1.0.0",
  "description": "Plugin personnalise pour mon equipe",
  "commands": [
    {
      "name": "custom:audit",
      "description": "Audit personnalise",
      "file": "commands/audit.md"
    }
  ],
  "hooks": {
```



```

    "PostToolUse": [
      {
        "matcher": "Write",
        "command": "hooks/post-write.sh"
      }
    ]
  }
}

```

7. Task Management System (10min)

Qu'est-ce que le Task Management ?

Le **Task Management System** (v2.1.19+) permet à Claude de créer, suivre et gérer des tâches structurées avec dépendances. C'est l'outil idéal pour les opérations complexes multi-étapes.

Les 4 outils

Outil	Description
TaskCreate	Créer une tâche avec sujet, description et activeForm (spinner)
TaskGet	Récupérer les détails complets d'une tâche par son ID
TaskUpdate	Mettre à jour statut, sujet, description, dépendances
TaskList	Lister toutes les tâches avec leur statut et dépendances

Cycle de vie (v2.1.20+)

```

pending → in_progress → completed
                ↓
                deleted

```

Note (v2.1.20+) : Le statut de `deleted` permet de supprimer définitivement une tâche du système via `TaskUpdate`.

Task Tool Metrics (v2.1.30+)

Les résultats du Task tool incluent désormais des métriques d'exécution :

Metrique	Description
Token count	Tokens totaux consommés par le sous-agent
Tool uses	Nombre d'invocations d'outils pendant l'exécution
Duration	Durée totale d'exécution de la tâche

Ces metriques permettent de monitorer le coût par tâche, identifier les opérations coûteuses et optimiser la distribution des tâches parallèles.

Gestion des dépendances

Les tâches supportent `blocks` et `blockedBy` pour structurer les plans :

```
{  
  "taskId": "2",  
  "addBlockedBy": ["1"]  
}
```

Une tâche bloquée ne peut démarrer que lorsque ses dépendances sont résolues.

Cas d'usage

- **Planification multi-étapes** : décomposer une feature en tâches ordonnées
- **Suivi de progression** : visualiser l'avancement en temps réel via `TaskList`
- **Orchestration** : identifier les tâches exécutables en parallèle vs séquentiel
- **Integration Ralph** : le Sprint Conductor utilise le Task Management pour le suivi

8. File Operation Tools vs Bash (5min)

Pourquoi préférer les outils natifs ?

Depuis la v2.1.21, Claude préfère les outils de fichiers natifs aux équivalents bash. Cette approche offre plusieurs avantages :

Avantage	Description
Fiabilité	Meilleure gestion des erreurs et des cas limites
Performance	Moins de latence que les commandes shell
Tracabilité	Meilleure visibilité dans les logs et hooks

Avantage	Description
Securite	Evite les injections de commandes

Mapping des outils

Tache	Outil natif	Eviter
Lire un fichier	Read	cat, head, tail
Modifier un fichier	Edit	sed, awk
Ecrire un fichier	Write	echo >, cat <<EOF
Rechercher des fichiers	Glob	find, ls
Rechercher dans le contenu	Grep	grep, rg

PDF Page Range Support (v2.1.30+)

Le Read tool supporte désormais un parametre pages pour les fichiers PDF :

Fonctionnalite	Description
pages parametre	Specifier une plage de pages (ex : pages : "1-5")
Optimisation grands PDFs	Les PDFs >10 pages retournent une reference legere quand mentionnes via @
Max pages (v2.1.31+)	100 pages par requete
Max taille (v2.1.31+)	20MB par fichier

Note (v2.1.31+) : Les system prompts ont ete renforces pour guider encore plus fortement Claude vers les outils natifs. Claude utilise désormais systematiquement Read, Edit, Glob et Grep au lieu des equivalents bash.

Quand utiliser Bash ?

Bash reste pertinent pour : - Operations systeme (git, docker, npm, etc.) - Commandes composees avec pipes complexes - Outils specifiques sans equivalent natif

9. PR Integration (5min)

Qu'est-ce que PR Integration ?

Depuis la v2.1.27, Claude Code s'intègre avec les Pull Requests GitHub, permettant de reprendre des sessions liées à des PRs et d'afficher le statut en temps réel.

Utilisation

```
# Reprendre une session liée à une PR par numéro  
claude --from-pr 123
```

```
# Reprendre une session liée à une PR par URL  
claude --from-pr https://github.com/org/repo/pull/123
```

Auto-link

Lorsque vous créez une PR via `gh pr create` pendant une session Claude Code, la session est automatiquement liée à cette PR.

Indicateurs de statut

Le footer de Claude Code affiche le statut de la PR liée :

Statut	Affichage
Approuvée	approved
En attente	pending
Changements demandés	changes requested
Brouillon	draft
Fusionnée	merged

10. Background Agents et Permissions (5min)

Permission Prompting (v2.1.20+)

Les agents en arrière-plan demandent maintenant les permissions **avant** le lancement, évitant les blocages en cours d'exécution.

Comportement

Launching background task: "Analyze and fix code"

This task will need permissions for:

- Read (all files)
- Edit (src/**)
- Bash (npm run lint:fix)

Approve all? [y/N/select]

Options de reponse

Option	Action
y	Approuve toutes les permissions demandees
N	Refuse et annule le lancement
select	Choisir les permissions individuellement

Avantages

- **Pas de blocage mid-execution** : toutes les permissions sont resolues au demarrage
- **Visibilite** : vous savez exactement ce que l'agent va faire
- **Controle granulaire** : possibilite de refuser certaines permissions

12. Personnalisation avancee (5min)

spinnerVerbs (v2.1.23+)

Claude Code permet de personnaliser les verbes affiches pendant l'execution des outils via la propriete `activeForm` des taches et la configuration `spinnerVerbs`.

```
{
  "spinnerVerbs": {
    "default": ["Thinking", "Processing"],
    "Edit": ["Editing", "Modifying"],
    "Bash": ["Running", "Executing"],
    "Read": ["Reading", "Loading"]
  }
}
```

Le lien avec `activeForm` dans `TaskCreate` permet d'afficher un texte personnalisé pendant l'exécution d'une tâche spécifique.

Settings Enhancement (v7.13.0)

Les paramètres de configuration ont été enrichis :

```
{
  "plansDirectory": ".claude/plans",
  "permissions": {
    "allow": [
      "Bash(npm *)", "Bash(pnpm *)", "Bash(yarn *)",
      "Bash/php *)", "Bash(flutter *)", "Bash(ng *)", "Bash(dotnet *)"
    ],
    "deny": [
      "Bash(chmod 777 *)",
      "Bash(*curl*|*sh*)",
      "Write(*credentials*)"
    ]
  }
}
```

- `plansDirectory` : répertoire dédié pour les plans généraux
- Permissions étendues pour les gestionnaires de paquets multi-technologie
- Deny rules pour bloquer les opérations dangereuses

11. Agent Teams (v2.1.32+ Research Preview) (10min)

Qu'est-ce qu'Agent Teams ?

Agent Teams permet à plusieurs agents Claude de travailler simultanément sur des tâches coordonnées. C'est un système de coordination multi-agents avec tâches partagées.

Activation

```
export CLAUDE_CODE_EXPERIMENTAL_AGENT_TEAMS=1
```

Outils disponibles

Outil	Description
Teammate	Créer une équipe (<code>spawnTeam</code>), nettoyer (<code>cleanup</code>)

Outil	Description
SendMessage	Envoyer des messages entre agents (message, broadcast, shutdown)
TaskCreate/Update/List/Get	Tâches partagées entre tous les agents de l'équipe

Fonctionnement

1. **Créer une équipe** : `Teammate.spawnTeam("mon-equipe")`
2. **Créer des tâches** : `TaskCreate` pour définir les tâches à réaliser
3. **Lancer des coéquipiers** : Spawn d'agents spécialisés via `Task tool` avec `team_name`
4. **Coordination** : Les agents communiquent via `SendMessage` et partagent la liste de tâches
5. **Completion** : Les agents marquent les tâches comme terminées et passent aux suivantes

Modes d'affichage

Mode	Navigation	Environnement
In-process	Shift+Up/Down	Terminal standard
Split panes	Panneau par agent	tmux / iTerm2
Delegate	Shift+Tab	Basculer entre agents

Cas d'usage

- **Refactoring parallèle** : Plusieurs agents modifient des modules indépendants
- **Recherche + implementation** : Un agent explore, un autre implémente
- **Sprint autonome** : Plusieurs stories traitées simultanément

Claude Opus 4.6 (v2.1.32+)

Le nouveau modèle flagship avec des capacités étendues :

Caractéristique	Valeur
Model ID	<code>claude-opus-4-6</code>
Contexte	200K standard, 1M beta
Output max	128K tokens
Adaptive thinking	Niveaux : low, medium, high, max

Points Cles a Retenir

1. **13 Hooks** : PreToolUse, PostToolUse, PostToolUseFailure, PermissionRequest, UserPrompt-Submit, Stop, SubagentStop, SubagentStart, Notification, PreCompact, SessionStart, SessionEnd, Setup - attention aux faux evenements dans certaines docs
2. **Ralph Wiggum** : Boucle IA continue avec DoD validators et circuit breaker adaptatif
3. **Sprint automation** : Ralph + BMAD commands + Agent Teams pour sprints paralleles
4. **Extended Thinking** : Hierarchie think < think hard < think harder < ultrathink
5. **MCP** : Model Context Protocol pour connecter des outils externes (bases de donnees, APIs, navigateur)
6. **QA Recette** : Tests d'acceptance automatises via Chrome avec la Regle d'Or (un bug corrige ne doit jamais reapparaître)
7. **Permissions** : Systeme 3-tier Deny > Allow > Ask avec wildcards et configuration multi-niveaux
8. **Task Management** : TaskCreate/Get/Update/List avec statut de leted (v2.1.20+)
9. **Hook Scripts Claude-Craft** : post-tool-failure.sh, pre-compact.sh, session-end.sh
10. **File Tools vs Bash** : Preferer Read/Edit/Write aux equivalents bash (v2.1.21+)
11. **PR Integration** : --from-pr et indicateurs de statut PR (v2.1.27+)
12. **Background Agent Permissions** : Demande des permissions avant lancement (v2.1.20+)
13. **PDF Page Range** : Parametre pages pour Read tool sur PDFs, ref legere >10 pages (v2.1.30+)
14. **Task Tool Metrics** : Token count, tool uses, duration dans resultats Task (v2.1.30+)
15. **OAuth MCP** : --client-id / --client-secret pour serveurs MCP sans DCR (v2.1.30+)
16. **Session Resume Hint** : Hint de reprise de session affiche a la sortie de Claude Code (v2.1.31+)
17. **PDF Limits** : Limites reelles clarifiees - max 100 pages, max 20MB (v2.1.31+)
18. **Claude Opus 4.6** : Nouveau modele flagship - 200K context (1M beta), 128K output, adaptive thinking (v2.1.32+)
19. **Agent Teams** : Coordination multi-agents avec Teammate/SendMessage, taches partagees (v2.1.32+ Research Preview)
20. **Automatic Memory** : Enregistrement auto de la memoire de session apres ~10K tokens (v2.1.32+)
21. **Skill Budget Scaling** : Budget skills = 2% de la fenetre de contexte du modele (v2.1.32+)
22. **Fast Mode** : / fast pour Opus 4.6 jusqu'a 2.5x plus rapide (v2.1.36+)
23. **Skills Directory Protection** : Ecritures dans .claude/skills bloques en sandbox (v2.1.45+)

Duree estimee : 1h45 **Prochain module** : Mise en Pratique Collective